SNPRINTF

Be careful with string formatting operations.

Sean Barnum, Cigital, Inc. [vita[1]]

2007-04-16

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 7861 bytes

Attack Category	• Malicious Input
Vulnerability Category	• Buffer Overflow • Format string • No Null Termination
Software Context	• String Formatting
Location	• stdio.h
Description	Writes into the character string str the result of formatting according to the string format (identical to printf format) the arguments following the string format. The string str is truncated to at most size-1 characters and '\0' is added at the end of the string. Returns the number of characters that would have been written if the string str were unlimited. Unfortunately, snprintf()'s variants have additional problems and are thus EXTREMELY unportable. Officially, snprintf() is not a standard C function in the ISO 1990 (ANSI 1989) standard, though sprintf() is, so not all systems include snprintf(). Even worse, some systems' snprintf() do not actually protect against buffer overflows; they just call sprintf directly. Old versions of Linux's libc4 depended on a "libbsd" with serious security shortcomings, and apparently some old HP systems did the same. Linux's current version of snprintf is known to work correctly, that is, it does actually respect the boundary requested. The return value of snprintf() varies as well; the Single Unix Specification (SUS) version 2 and the C99 standard differ on what is returned by snprintf(). Finally, it appears that at least some versions of snprintf don't guarantee that its string will end in NULL; if the string is too long, it won't include NULL at all. Note that the glib library (the basis of GTK, and not the same as the GNU C library glibc) has a g_snprintf(), which has a consistent return semantic, always null-terminates,

1. http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

and most importantly always respects the buffer length.

In general, snprintf
1. can return a negative number if the buffer is too small
2. can return the number of bytes that it should have written
3. might not null terminate the string
4. might null terminate the string

Note: snprintf() is a good substitute for strcat() and similar string concatenation jobs because it takes a maximum buffer size, not just a limit on characters to write (i.e. the user does not need to keep track of it).

APIs

Function Name	Comments
_snprintf	fmt: 2; src 3 variable;
_sntprintf	fmt: 2; src 3 variable;
_snwprintf	fmt: 1; src: 3 variable;
_vsnprintf	fmt: 2; src 3 variable;
_vsntprintf	fmt: 2; src 3 variable;
_vsnwprintf	fmt: 2; src 3 variable;
snprintf	fmt: 2; src: 3 variable;
snwprintf	fmt: 1; src: 3 variable;
vsnprintf	fmt: 1; src: 3 variable;
wnsprintf	fmt: 1; src: 3 variable;
wvnsprintf	fmt: 1; src: 3 variable;

Method of Attack

An attacker could potentially input an excessively long string that when used by snprintf() could result in a buffer overflow.

The snprintf() and _snprintf() functions are generally non-portable.

Exception Criteria

Solutions

Solution Applicability	Solution Description	Solution Efficacy
	Portability is generally sacrificed if snprintf is used. 1. Thorough analysis (string testing) of buffer overflow and all error scenarios and	

		verification of return values must be done. 2. snprintf() CAN BE safer than sprintf, depending on how snprintf() was implemented. The most serious problem with snprintf() can occur when snprintf() is implemented simply by calling sprintf(). Therefore the best solution for protecting snprintf() (in a generally portable manner) is to perform the bounds checking solutions as described in sprintf(). (See sprintf() rule.)	

Signature Details	int snprintf(char *str, size_t size, const char *format, ...);

Examples of Incorrect Code	`/* Again, the real problem with snprintf() has to do with portability. */`
	`/* Generally snprintf can be safer than sprintf(), but it is not guaranteed */`
	`/* based on the implementation. Therefore any use of snprintf() that doesn't */`
	`/* incorporate application based bound checks are thus vulnerable */`
	
	`[...]`
	
	`snprintf(dst, sizeof(dst) - 1, "%s", src) /* and see below */`
	
	`[...]`

Examples of Corrected Code	`/* See sprintf() rule */`
	`/* and additionally */`
	`if (snprintf(dst, sizeof(dst) - 1, "%s", src) > sizeof(dst) - 1) {`
	`/* Overflow */`
	`...`
	`}`

Source References	• Howard, Michael & LeBlanc, David C. *Writing Secure Code, 2nd ed*. Redmond, WA: Microsoft Press, 2002, ISBN: 0735617228. • http://howtos.linux.com/howtos/Secure-Programs-HOWTO/dangers-c.shtml
Recommended Resource	
Discriminant Set	**Operating Systems**: • Windows • UNIX • Linux • HP-UX **Languages**: • C • C++

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com